Application for United States Letters Patent

for

# MEMORY MANAGEMENT SYSTEM AND METHOD PROVIDING LINEAR ADDRESS BASED MEMORY ACCESS SECURITY

by

**Brian C. Barnes**

**Geoffrey S. Strongin**

**Rodney W. Schmidt**

EXPRESS MAIL MAILING LABEL

NUMBER EL798365109US

DATE OF DEPOSIT 13 November 2001

I hereby certify that this paper or fee is being deposited with the United States Postal Service "EXPRESS MAIL POST OFFICE TO ADDRESSEE" service under 37 C.F.R. 1.10 on the date indicated above and is addressed to Assistant Commissioner for Patents, Washington D.C. 20231.

Sharon Hart

SIGNATURE

# MEMORY MANAGEMENT SYSTEM AND METHOD PROVIDING LINEAR ADDRESS BASED MEMORY ACCESS SECURITY

## BACKGROUND OF THE INVENTION

### 1. REFERENCES

This patent application is related to a co-pending patent application serial no. _____ (Attorney Reference Number 2000.056600/TT4086) entitled "Memory Management System And Method Providing Physical Address Based Memory Access Security" by Brian C. Barnes, Geoffrey S. Strongin, and Rodney W. Schmidt, filed on the same day as the present patent application.

### 2. FIELD OF THE INVENTION

This invention relates generally to memory management systems and methods, and, more particularly, to memory management systems and methods that provide protection for data stored within a memory.

### 3. DESCRIPTION OF THE RELATED ART

A typical computer system includes a memory hierarchy to obtain a relatively high level of performance at a relatively low cost. Instructions of several different software programs are typically stored on a relatively large but slow non-volatile storage unit (e.g., a disk drive unit). When a user selects one of the programs for execution, the instructions of the selected program are copied into a main memory unit (e.g., random access memory (RAM)), and a central processing unit (CPU) obtains the instructions of the selected program from the main memory unit. A well-known virtual memory management technique allows the CPU to access data structures larger in size than that of the main memory unit by storing only a portion of the data structures within the main memory unit at any given time.

Remainders of the data structures are stored within the relatively large but slow non-volatile storage unit, and are copied into the main memory unit only when needed.

Virtual memory is typically implemented by dividing an address space of the CPU into multiple blocks called page frames or "pages." Only data corresponding to a portion of the pages is stored within the main memory unit at any given time. When the CPU generates an address within a given page, and a copy of that page is not located within the main memory unit, the required page of data is copied from the relatively large but slow non-volatile storage unit into the main memory unit. In the process, another page of data may be copied from the main memory unit to the non-volatile storage unit to make room for the required page.

The popular 80x86 (x86) processor architecture includes specialized hardware elements to support a protected virtual address mode (i.e., a protected mode). Figs. 1-3 will now be used to describe how an x86 processor implements both virtual memory and memory protection features. Fig. 1 is a diagram of a well-known linear-to-physical address translation mechanism 100 of the x86 processor architecture. An address translation mechanism 100 is embodied within an x86 processor, and involves a linear address 102 produced within the x86 processor, a page table directory (i.e., a page directory) 104, multiple page tables including a page table 106, multiple page frames including a page frame 108, and a control register (CR3) 110. The page directory 104 and the multiple page tables are paged memory data structures created and maintained by operating system software (i.e., an operating system). The page directory 104 is always located within the memory (e.g., the main memory unit). For simplicity, the page table 106 and the page frame 108 will also be assumed to reside in the memory.

As indicated in Fig. 1, the linear address 102 is divided into three portions to accomplish the linear-to-physical address translation. The highest ordered bits of the CR3

110 are used to store a page directory base register. The page directory base register is a base address of a memory page containing the page directory 104. The page directory 104 includes multiple page directory entries, including a page directory entry 112. An upper "directory index" portion of the linear address 102, including the highest ordered or most significant bits of the linear address 102, is used as an index into the page directory 104. The page directory entry 112 is selected from within the page directory 104 using the page directory base address of the CR3 110 and the upper "directory index" portion of the linear address 102.

Fig. 2 is a diagram of a page directory entry format 200 of the x86 processor architecture. As indicated in Fig. 2, the highest ordered (i.e., most significant) bits of a given page directory entry contain a page table base address, where the page table base address is a base address of a memory page containing a corresponding page table. The page table base address of the page directory entry 112 is used to select the corresponding page table 106.

Referring back to Fig. 1, the page table 106 includes multiple page table entries, including a page table entry 114. A middle "table index" portion of the linear address 102 is used as an index into the page table 106, thereby selecting the page table entry 114. Fig. 3 is a diagram of a page table entry format 300 of the x86 processor architecture. As indicated in Fig. 3, the highest ordered (i.e., most significant) bits of a given page table entry contain a page frame base address, where the page frame base address is a base address of a corresponding page frame.

Referring again to Fig. 1, the page frame base address of the page table entry 114 is used to select the corresponding page frame 108. The page frame 108 includes multiple memory locations. A lower or "offset" portion of the linear address 102 is used as an index into the page frame 108. When combined, the page frame base address of the page table entry 114 and the offset portion of the linear address 102 produce the physical address

corresponding to the linear address 102, and indicate a memory location 116 within the page frame 108. The memory location 116 has the physical address resulting from the linear-to-physical address translation.

Regarding the memory protection features, the page directory entry format 200 of Fig. 2 and the page table entry format 300 of Fig. 3 include a user/supervisor (U/S) bit and a read/write (R/W) bit. The contents of the U/S and R/W bits are used by the operating system to protect corresponding page frames (i.e., memory pages) from unauthorized access. U/S=0 is used to denote operating system memory pages, and corresponds to a "supervisor" level of the operating system. The supervisor level of the operating system corresponds to a current privilege level 0 (CPL0) of software programs and routines executed by the x86 processor. U/S=1 is used to indicate user memory pages, and corresponds to a "user" level of the operating system. The user level of the operating system corresponds to CPL3 of the x86 processor. (The user level may also correspond to CPL1 and/or CPL2 of the x86 processor.)

The R/W bit is used to indicate types of accesses allowed to the corresponding memory page. R/W=0 indicates the only read accesses are allowed to the corresponding memory page (i.e., the corresponding memory page is "read-only"). R/W=1 indicates that both read and write accesses are allowed to the corresponding memory page (i.e., the corresponding memory page is "read-write").

During the linear-to-physical address translation operation of Fig. 1, the contents of the U/S bits of the page directory entry 112 and the page table entry 114, corresponding to the page frame 108, are logically ANDed to determine if the access to the page frame 108 is authorized. Similarly, the contents of the R/W bits of the page directory entry 112 and the page table entry 114 are logically ANDed to determine if the access to the page frame 108 is authorized. If the logical combinations of the U/S and R/W bits indicate the access to the page frame 108 is authorized, the memory location 116 is accessed using the physical

address. On the other hand, if the logical combinations of the U/S and R/W bits indicate that the access to the page frame 108 is not authorized, the memory location 116 is not accessed, and a protection fault indication is signaled.

Unfortunately, the above described memory protection mechanisms of the x86 processor architecture are not sufficient to protect data stored in the memory. For example, any software program or routine executing at the supervisor level (e.g., having a CPL of 0) can access any portion of the memory, and can modify (i.e., write to) any portion of the memory that is not marked "read-only" (R/W=0). In addition, by virtue of executing at the supervisor level, the software program or routine can change the attributes (i.e., the U/S and R/W bits) of any portion of the memory. The software program or routine can thus change any portion of the memory marked "read-only" to "read-write" (R/W=1), and then proceed to modify that portion of the memory.

The present invention is directed to a method that may solve, or at least reduce, some or all of the aforementioned problems, and systems incorporating the method.

## SUMMARY OF THE INVENTION

A memory management unit is disclosed for managing a memory storing data arranged within a plurality of memory pages. The memory management unit includes a security check unit receiving a linear address generated during execution of a current instruction. The linear address has a corresponding physical address residing within a selected memory page. The security check unit uses the linear address to access one or more security attribute data structures located in the memory to obtain a security attribute of the selected memory page. The security check unit compares a numerical value conveyed by a security attribute of the current instruction to a numerical value conveyed by the security attribute of the selected memory page, and produces an output signal dependent upon a result of the comparison. The memory management unit accesses the selected memory page dependent upon the output signal.

A central processing unit (CPU) is described including an execution unit and the above described memory management unit (MMU). The execution unit fetches instructions from a memory and executes the instructions. The MMU manages the memory, and is configurable to manage the memory such that the memory stores data arranged within multiple memory pages. The MMU includes the above described security check unit. A computer system is described including the memory, the CPU, and the MMU.

A method is described for providing access security for a memory used to store data arranged within a plurality of memory pages. The method includes receiving a linear address produced during execution of an instruction, and a security attribute of the instruction, wherein the instruction resides in a first memory page. The linear address is used to access one or more paged memory data structures located in the memory to obtain a base address of a selected memory page, and security attributes of the selected memory page. If the security attribute of the instruction (e.g., a current privilege level or CPL of the instruction as defined by the x86 processor architecture) and the security attributes of the selected memory page (e.g., x86 U/S and R/W bits of the selected memory page) indicate the access is authorized, the base address of the selected memory page is combined with an offset to produce a physical address within the selected memory page. If the security attribute of the instruction and the security attributes of the selected memory page indicate the access is not authorized, a fault signal (e.g., an x86 GPF signal) is generated.

The linear address is also used to access one or more security attribute data structures located in the memory to obtain an additional security attribute of the first memory page, and an additional security attribute of the selected memory page. A numerical value conveyed by an additional security attribute of the first memory page is compared to a numerical value conveyed by the additional security attribute of selected memory page. The selected memory page is accessed dependent upon a result of the comparing of the numerical values conveyed

by the security attribute of the first memory page and the additional security attribute of selected memory page.

## BRIEF DESCRIPTION OF THE DRAWINGS

The invention may be understood by reference to the following description taken in conjunction with the accompanying drawings, in which like reference numerals identify similar elements, and in which:

Fig. 1 is a diagram of a well-known linear-to-physical address translation mechanism of the x86 processor architecture;

Fig. 2 is a diagram of a page directory entry format of the x86 processor architecture;

Fig. 3 is a diagram of a page table entry format of the x86 processor architecture;

Fig. 4 is a diagram of one embodiment of a computer system including a CPU, wherein the CPU includes a CPU security check unit (SCU);

Fig. 5 is a diagram illustrating relationships between various hardware and software components of the computer system of Fig. 4;

Fig. 6 is a diagram of one embodiment of the CPU of the computer system of Fig. 4, wherein the CPU includes a memory management unit (MMU);

Fig. 7 is a diagram of one embodiment of the MMU of Fig. 6, wherein the MMU includes a paging unit coupled to the CPU SCU;

Fig. 8 is a diagram of one embodiment of the CPU SCU of Fig. 7;

Fig. 9 is a diagram of one embodiment of a mechanism for accessing a security attribute table (SAT) entry of a given memory page to obtain additional security information of the given memory page;

Fig. 10 is a diagram of one embodiment of a SAT default register;

Fig. 11 is a diagram of one embodiment of a SAT directory entry format;

Fig. 12 is a diagram of one embodiment of a SAT entry format; and

Figs. 13A and 13B in combination form a flow chart of one embodiment of a method for managing a memory used to store data arranged within multiple memory pages.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof have been shown by way of example in the drawings and are herein described in detail. It should be understood, however, that the description herein of specific embodiments is not intended to limit the invention to the particular forms disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the invention as defined by the appended claims.

## DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Illustrative embodiments of the invention are described below. In the interest of clarity, not all features of an actual implementation are described in this specification. It will, of course, be appreciated that in the development of any such actual embodiment, numerous implementation-specific decisions must be made to achieve the developers' specific goals, such as compliance with system-related and business-related constraints, which will vary from one implementation to another. Moreover, it will be appreciated that such a development effort might be complex and time-consuming, but would nevertheless be a routine undertaking for those of ordinary skill in the art having the benefit of this disclosure.

Fig. 4 is a diagram of one embodiment of a computer system 400 including a CPU 402, a system or "host" bridge 404, a memory 406, a first device bus 408 (e.g., a peripheral component interconnect or PCI bus), a device bus bridge 410, a second device bus 412 (e.g., an industry standard architecture or ISA bus), and four device hardware units 414A-414D. The host bridge 404 is coupled to the CPU 402, the memory 406, and the first device bus 408. The host bridge 404 translates signals between the CPU 402 and the first device bus 408, and operably couples the memory 406 to the CPU 402 and to the first device bus 408. The device bus bridge 410 is coupled between the first device bus 408 and the second device bus 412, and translates signals between the first device bus 408 and the second device bus

412. In the embodiment of Fig. 4, the device hardware units 414A and 414B are coupled to the first device bus 408, and the device hardware units 414C and 414D are coupled to the second device bus 412. One or more of the device hardware units 414A-414D may be, for example, storage devices (e.g., hard disk drives, floppy drives, and CD-ROM drives), communication devices (e.g., modems and network adapters), or input/output devices (e.g., video devices, audio devices, and printers).

In the embodiment of Fig. 4, the CPU 402 includes a CPU security check unit (SCU) 416. As will be described in detail below, the CPU SCU 416 protects the memory 406 from unauthorized accesses generated by the CPU 402. It is noted that in other embodiments, the host bridge 404 may be part of the CPU 402 as indicated in Fig. 4.

Fig. 5 is a diagram illustrating relationships between various hardware and software components of the computer system 400 of Fig. 4. In the embodiment of Fig. 5, multiple application programs 500, an operating system 502, a security kernel 504, and device drivers 506A-506D are stored in the memory 406. The application programs 500, the operating system 502, the security kernel 504, and the device drivers 506A-506D include instructions executed by the CPU 402. The operating system 502 provides a user interface and software "platform" on top of which the application programs 500 run. The operating system 502 may also provide, for example, basic support functions, including file system management, process management, and input/output (I/O) control.

The operating system 502 may also provide basic security functions. For example, the CPU 402 (Fig. 4) may be an x86 processor that executes instructions of the x86 instruction set. In this situation, the CPU 402 may include specialized hardware elements to provide both virtual memory and physical memory protection features in the protected mode as described above. The operating system 502 may be, for example, one of the Windows® family of operating systems (Microsoft Corp., Redmond, WA) that operates the CPU 402 in

the protected mode, and uses the specialized hardware elements of the CPU 402 to provide both virtual memory and memory protection in the protected mode.

As will be described in more detail below, the security kernel 504 provides additional security functions above the security functions provided by the operating system 502 to protect data stored in the memory 406 from unauthorized access. As indicated in Fig. 5, the security kernel 504 is coupled to the CPU SCU 416. As will be described in detail below, the CPU SCU 416 monitors all software-initiated accesses to the memory 406, and allows only authorized accesses to the memory 406.

In the embodiment of Fig. 5, the device drivers 506A-506D are operationally associated with, and coupled to, the respective corresponding device hardware units 414A-414D. The device hardware units 414A and 414D may be, for example, "secure" devices, and the corresponding device drivers 506A and 506D may be "secure" device drivers. The security kernel 504 is coupled between the operating system 502 and the secure device drivers 506A and 506D, and may monitor all accesses by the application programs 500 and the operating system 502 to secure the device drivers 506A and 506D and the corresponding secure devices 414A and 414D. The security kernel 504 may prevent unauthorized accesses to the secure device drivers 506A and 506D and the corresponding secure devices 414A and 414D by the application programs 500 and the operating system 502. The device drivers 506B and 506C, on the other hand, may be "non-secure" device drivers, and the corresponding device hardware units 414B and 414C may be "non-secure" device hardware units. The device drivers 506B and 506C and the corresponding device hardware units 414B and 414C may be, for example, "legacy" device drivers and device hardware units.

It is noted that in other embodiments, the security kernel 504 may be part of the operating system 502. In yet other embodiments, the security kernel 504, the device drivers

506A and 506D, and/or the device drivers 506B and 506C may be part of the operating system 502.

Fig. 6 is a diagram of one embodiment of the CPU 402 of the computer system 400 of Fig. 4. In the embodiment of Fig. 6, the CPU 402 includes an execution unit 600, a memory management unit (MMU) 602, a cache unit 604, a bus interface unit (BIU) 606, a set of control registers 608, and a set of secure execution mode (SEM) registers 610. The CPU SCU 416 is located within the MMU 602. As will be described in detail below, the set of SEM registers 610 are used to implement a secure execution mode (SEM) within the computer system 400 of Fig. 4, and the operation of the CPU SCU 416 is governed by the contents of the set of SEM registers 610. SEM registers 610 are accessed (i.e., written to and/or read from) by the security kernel 504 (Fig. 5). The computer system 400 of Fig. 4 may, for example, operate in the SEM when: (i) THE CPU 402 is an x86 processor operating in the x86 protected mode, (ii) memory paging is enabled, and (iii) the contents of SEM registers 610 specify SEM operation.

In general, the contents of the set of control registers 608 govern operation of the CPU 402. Accordingly, the contents of the set of control registers 608 govern operation of the execution unit 600, the MMU 602, the cache unit 604, and/or the BIU 606. The set of control registers 608 may include, for example, the multiple control registers of the x86 processor architecture.

The execution unit 600 of the CPU 402 fetches instructions (e.g., x86 instructions) and data, executes the fetched instructions, and generates signals (e.g., address, data, and control signals) during instruction execution. The execution unit 600 is coupled to the cache unit 604, and may receive instructions from the memory 406 (Fig. 4) via the cache unit 604 and the BIU 606.

The memory 406 (Fig. 4) of the computer system 400 includes multiple memory locations, each having a unique physical address. When operating in protected mode with paging enabled, an address space of the CPU 402 is divided into multiple blocks called page frames or "pages." As described above, only data corresponding to a portion of the pages is stored within the memory 406 at any given time. In the embodiment of Fig. 6, address signals generated by the execution unit 600 during instruction execution represent segmented (i.e., "logical") addresses. As described below, the MMU 602 translates the segmented addresses generated by the execution unit 600 to corresponding physical addresses of the memory 406. The MMU 602 provides the physical addresses to the cache unit 604. The cache unit 604 is a relatively small storage unit used to store instructions and data recently fetched by the execution unit 600. The BIU 606 is coupled between the cache unit 604 and the host bridge 404, and is used to fetch instructions and data not present in the cache unit 604 from the memory 406 via the host bridge 404.

Fig. 7 is a diagram of one embodiment of the MMU 602 of Fig. 6. In the embodiment of Fig. 7, the MMU 602 includes a segmentation unit 700, a paging unit 702, the CPU SCU 416, and selection logic 704 for selecting between outputs of the segmentation unit 700 and the paging unit 702 to produce a physical address. As indicated in Fig. 7, the segmentation unit 700 receives a segmented address from the execution unit 600 and uses a well-know segmented-to-linear address translation mechanism of the x86 processor architecture to produce a corresponding linear address at an output. As indicated in Fig. 7, when enabled by a "PAGING" signal, the paging unit 702 receives the linear addresses produced by the segmentation unit 700 and produces corresponding physical addresses. The PAGING signal may mirror the paging flag (PG) bit in a control register 0 (CR0) of the x86 processor architecture and of the set of control registers 608 (Fig. 6). When the PAGING signal is

deasserted, memory paging is not enabled, and the selection logic 704 produces the linear address received from the segmentation unit 700 as the physical address.

When the PAGING signal is asserted, memory paging is enabled, and the paging unit 702 translates the linear address received from the segmentation unit 700 to a corresponding physical address using the above described linear-to-physical address translation mechanism 100 of the x86 processor architecture (Fig. 1). As described above, during the linear-to-physical address translation operation, the contents of the U/S bits of the selected page directory entry and the selected page table entry are logically ANDed to determine if the access to a page frame is authorized. Similarly, the contents of the R/W bits of the selected page directory entry and the selected page table entry are logically ANDed to determine if the access to the page frame is authorized. If the logical combinations of the U/S and R/W bits indicate the access to the page frame is authorized, the paging unit 702 produces the physical address resulting from the linear-to-physical address translation operation. The selection logic 704 receives the physical address produced by the paging unit 702, produces the physical address received from the paging unit 702 as the physical address, and provides the physical address to the cache unit 604.

On the other hand, if the logical combinations of the U/S and R/W bits indicate the access to the page frame 108 is not authorized, the paging unit 702 does not produce a physical address during the linear-to-physical address translation operation. Instead, the paging unit 702 asserts a general protection fault (GPF) signal, and the MMU 602 forwards the GPF signal to the execution unit 600. In response to the GPF signal, the execution unit 600 may execute an exception handler routine, and may ultimately halt the execution of one of the application programs 500 (Fig. 5) running when the GPF signal was asserted. The paging unit 702 may also include a translation lookaside buffer (TLB) for storing a relatively small number of recently determined linear-to-physical address translations.

Fig. 8 is a diagram of one embodiment of the CPU SCU 416 of Fig. 7. In the embodiment of Fig. 8, the CPU SCU 416 includes security check logic 800 coupled to the set of SEM registers 610 (Fig. 6) and a security attribute table (SAT) entry buffer 802. As described below, SAT entries include additional security information above the U/S and R/W bits of page directory and page table entries corresponding to memory pages. The security check logic 800 uses the additional security information stored within a given SAT entry to prevent unauthorized software-initiated accesses to the corresponding memory page. The SAT entry buffer 802 is used to store a relatively small number of SAT entries of recently accessed memory pages.

As described above, the set of SEM registers 610 are used to implement a secure execution mode (SEM) within the computer system 400 of Fig. 4. The contents of the set of SEM registers 610 govern the operation of the CPU SCU 416. The security check logic 800 receives information to be stored in the SAT entry buffer 802 from the MMU 602 via a communication bus indicated in Fig. 8. The security check logic 800 also receives a physical address produced by the paging unit 702.

Figs. 9-11 will now be used to describe how additional security information of a selected memory page is obtained within the computer system 400 of Fig. 4. Fig. 9 is a diagram of one embodiment of a mechanism 900 for accessing a SAT entry of the selected memory page to obtain additional security information of the selected memory page. The mechanism 900 of Fig. 9 may be embodied within the security check logic 800 of Fig. 8, and may be implemented when the computer system 400 of Fig. 4 is operating in the SEM. The mechanism 900 involves a linear address 902 produced by the segmentation unit 700 (Fig. 7) during execution of an instruction, an SAT directory 904, multiple SATs including a SAT 906, and a SAT base address register 908 of the set of SEM registers 610. The SAT directory 904 and the multiple SATs, including the SAT 906, are SEM data structures created and

maintained by the security kernel 504 (Fig. 5). As described below, the SAT directory 904 (when present) and any needed SAT are copied into the memory 406 before being accessed.

The SAT base address register 908 includes a present (P) bit which indicates the presence of a valid SAT directory base address within the SAT base address register 908. The highest ordered (i.e., most significant) bits of the SAT base address register 908 are reserved for the SAT directory base address. The SAT directory base address is a base address of a memory page containing the SAT directory 904. If P=1, the SAT directory base address is valid, and SAT tables specify the security attributes of memory pages. If P=0, the SAT directory base address is not valid, no SAT tables exist, and security attributes of memory pages are determined by a SAT default register.

Fig. 10 is a diagram of one embodiment of the SAT default register 1000. In the embodiment of Fig. 10, the SAT default register 1000 includes a secure context identification (SCID) field. The SCID field includes multiple bit positions of the SAT default register 1000. The bit positions form a binary representation of an SCID value, wherein the SCID value is an integer value greater than or equal to 0. The SCID value indicates a default security context level for memory pages in the computer system 400. For example, where the SCID field includes $n$ bits of the SAT default register 1000, a given memory page may occupy one of $2^n$ possible security context levels in the computer system 400.

The linear address 902 has a corresponding physical address residing in the selected memory page. The corresponding physical address is generated by the paging unit 702 (Fig. 7) using the address translation mechanism 100 of Fig. 1. Referring back to Fig. 9 and assuming the P bit of SAT base address register 908 is a '1', the linear address 902 produced by the segmentation unit 700 (Fig. 7) is divided into three portions to access the SAT entry of the selected memory page. As described above, the SAT directory base address of the SAT base address register 908 is the base address of the memory page containing the SAT

directory 904. The SAT directory 904 includes multiple SAT directory entries, including a SAT directory entry 910. Each SAT directory entry may have a corresponding SAT in the memory 406. An "upper" portion of the linear address 902, including the highest ordered or most significant bits of the linear address 902, is used as an index into the SAT directory 904. The SAT directory entry 910 is selected from within the SAT directory 904 using the SAT directory base address of the SAT base address register 908 and the upper portion of the linear address 902.

Fig. 11 is a diagram of one embodiment of a SAT directory entry format 1100. In accordance with Fig. 11, each SAT directory entry includes a present (P) bit which indicates the presence of a valid SAT base address within the SAT directory entry. In the embodiment of Fig. 11, the highest ordered (i.e., the most significant) bits of each SAT directory entry are reserved for a SAT base address. The SAT base address is a base address of a memory page containing a corresponding SAT. If P=1, the SAT base address is valid, and the corresponding SAT is stored in the memory 406.

If P=0, the SAT base address is not valid, and the corresponding SAT does not exist in the memory 406 and must be copied into the memory 406 from a storage device (e.g., a disk drive). If P=0, the security check logic 800 may signal a page fault to logic within the paging unit 702, and the MMU 602 may forward the page fault signal to the execution unit 600 (Fig. 6). In response to the page fault signal, the execution unit 600 may execute a page fault handler routine which retrieves the needed SAT from the storage device, and store the needed SAT in the memory 406. After the needed SAT is stored in the memory 406, the P bit of the corresponding SAT directory entry is set to '1', and the mechanism 900 is continued. Alternately, if P=0, the contents of a SAT default base address field (not shown) of the SAT default register 1000 (Fig. 10) may be used as the SAT base address.

Referring back to Fig. 9, a "middle" portion of the linear address 902 is used as an index into the SAT 906. The SAT entry 906 is thus selected within the SAT 906 using the SAT base address of the SAT directory entry 910 and the middle portion of the linear address 902. Fig. 12 is a diagram of one embodiment of a SAT entry format 1200. In the embodiment of Fig. 12, each SAT entry includes a secure context identification (SCID) field. The SCID field includes multiple bit positions of the SAT entry. The bit positions form a binary representation of an SCID value, where the SCID value is an integer value greater than or equal to 0. The SCID value indicates a security context level of the selected memory page. For example, where the SCID field includes $n$ bits of the SAT entry, the selected memory page may occupy one of $2^n$ possible security context levels in the computer system 400.

The BIU 606 (Fig. 6) retrieves needed SEM data structure entries from the memory 406, and provides the SEM data structure entries to the MMU 602. Referring back to Fig. 8, the security check logic 800 receives SEM data structure entries from the MMU 602 and the paging unit 702 via the communication bus. As described above, the SAT entry buffer 802 is used to store a relatively small number of SAT entries of recently accessed memory pages. The security check logic 800 stores a given SAT entry in the SAT entry buffer 802, along with a "tag" portion of the corresponding physical address.

During a subsequent memory page access, the security check logic 800 may compare a "tag" portion of a physical address produced by the paging unit 702 to tag portions of physical addresses corresponding to SAT entries stored in the SAT entry buffer 802. If the tag portion of the physical address matches a tag portion of a physical address corresponding to a SAT entry stored in the SAT entry buffer 802, the security check logic 800 may access the SAT entry in the SAT entry buffer 802, eliminating the need to perform the process of Fig. 9 to obtain the SAT entry from the memory 406. The security kernel 504 (Fig. 5) modifies the contents of the SAT base address register 908 in the CPU 402 (e.g., during

context switches). In response to modifications of the SAT base address register 908, the security check logic 800 of the CPU SCU 416 may flush the SAT entry buffer 802.

In the embodiment of Fig. 7, the CPU SCU 416 is coupled to the paging unit 702. When the computer system 400 of Fig. 4 is operating in the SEM, the security check logic 800 receives the current privilege level (CPL) of the currently executing task (i.e., the currently executing instruction). The security check logic 800 also receives the page directory entry (PDE) U/S bit, the PDE R/W bit, the page table entry (PTE) U/S bit, and the PTE R/W bit of the selected memory page within which the corresponding physical address resides, from the paging unit 702. The security check logic 800 uses the above information, along with the SCID value of the SAT entry corresponding to the selected memory page, to determine if the memory 406 access is authorized.

The CPU 402 of Fig. 6 may be an x86 processor, and may include a code segment (CS) register, one of the 16-bit segment registers of the x86 processor architecture. Each segment register selects a block of memory (e.g., a segment of memory). In the protected mode with paging enabled, the CS register is loaded with a segment selector that indicates an executable block of the memory 406. The highest ordered (i.e., most significant) bits of the segment selector are used to store information indicating a block of memory (e.g., a segment of memory) including a next instruction to be executed by the execution unit 600 of the CPU 402 (Fig. 6). An instruction pointer (IP/EIP/RIP) register is used to store an offset into the block of memory (e.g., the segment of memory) indicated by the CS register. The CS:IP/EIP/RIP pair indicate a segmented address of the next instruction. The two lowest ordered (i.e., least significant) bits of the CS register are used to store a value indicating a current privilege level (CPL) of a task currently being executed by the execution unit 600 (i.e., the CPL of the current task).

Table 1 below illustrates exemplary rules for CPU-initiated (i.e., software-initiated) memory accesses when the computer system 400 of Fig. 4 is operating in the SEM. The CPU SCU 416 (Figs. 4-8) and the security kernel 504 (Fig. 5) work together to implement the rules of Table 1 when the computer system 400 of Fig. 4 is operating in the SEM to provide additional security for data stored in the memory 406 over and above data security provided by the operating system 502 (Fig. 5).

In Table 1 below, "CEI-SCID" represents the SCID value of the currently executing instruction, and "SMP-SCID" represents the SCID value of the selected memory page. The CEI-SCID value is the SCID value of the SAT entry corresponding to the memory page containing the currently executing instruction. In the embodiment of Table 1, lower SCID values are associated with higher security context levels. In other embodiments, higher SCID values may be associated with higher security context levels.

Table 1. Exemplary Rules For Software-Initiated Memory Accesses When The Computer System 400 Of Fig. 4 Is Operating In The SEM.

| SCID Comparison | CPL | Selected Memory Page U/S | Selected Memory Page R/W | Permitted Access | Remarks |
|---|---|---|---|---|---|
| CEI-SCID > SMP-SCID | X | X | X | None | SEM Security Exception asserted. |
| CEI-SCID <= SMP-SCID | 0 | X | 1(R/W) | R/W | Normal x86 protections apply: Full access granted. |

| | | | | | |
|---|---|---|---|---|---|
| CEI-SCID <= SMP-SCID | 0 | X | 0(R) | Read only | Normal x86 protections apply: Only read access. |
| CEI-SCID <= SMP-SCID | 3 | 0 | X | None | Normal x86 protections apply: GPF asserted. |
| CEI-SCID <= SMP-SCID | 3 | 1 | 1(R/W) | R/W | Normal x86 protections apply: Full access granted. |
| CEI-SCID <= SMP-SCID | 3 | 1 | 0(R) | Read only | Normal x86 protections apply: Only read access. |

In Table 1 above, the "CEI-SCID > SMP-SCID" condition is true when the integer value represented by the SCID value of the currently executing instruction is numerically greater than the integer value represented by the SCID value of the selected memory page. The "CEI-SCID <= SMP-SCID" condition is true when the integer value represented by the SCID value of the currently executing instruction is numerically less than or equal to the integer value represented by the SCID value of the selected memory page. The U/S bit of the selected memory page is the logical AND of the PDE U/S bit and the PTE U/S bit of the selected memory page. The R/W bit of the selected memory page is the logical AND of the PDE R/W bit and the PTE R/W bit of the selected memory page. The symbol "X" signifies a "don't care": the logical value may be either a '0' or a '1'.

Referring back to Fig. 8, the security check logic 800 of the CPU SCU 416 produces a general protection fault ("GPF") signal and a "SEM SECURITY EXCEPTION" signal, and

provides the GPF and the SEM SECURITY EXCEPTION signals to logic within the MMU 602. When the security check logic 800 asserts the GPF signal, the MMU 602 forwards the GPF signal to the execution unit 600 (Fig. 6). In response to the GPF signal, the execution unit 600 may use the well-known interrupt descriptor table (IDT) vectoring mechanism of the x86 processor architecture to access and execute a GPF handler routine.

When the security check logic 800 asserts the SEM SECURITY EXCEPTION signal, the MMU 602 forwards the SEM SECURITY EXCEPTION signal to the execution unit 600. Unlike normal processor exceptions which use the IDT vectoring mechanism of the x86 processor architecture, a different vectoring method may be used to handle SEM security exceptions. SEM security exceptions may be dispatched through a pair of registers (e.g., model specific registers or MSRs) similar to the way x86 "SYSENTER" and "SYSEXIT" instructions operate. The pair of registers may be "security exception entry point" registers, and may define a branch target address for instruction execution when a SEM security exception occurs. The security exception entry point registers may define the code segment (CS), then instruction pointer (IP, or the 64-bit version RIP), stack segment (SS), and the stack pointer (SP, or the 64-bit version RSP) values to be used on entry to a SEM security exception handler. Under software control, the execution unit 600 (Fig. 6) may push the previous SS, SP/RSP, EFLAGS, CS, and IP/RIP values onto a new stack to indicate where the exception occurred. In addition, the execution unit 600 may push an error code onto the stack. It is noted that a normal return from interrupt (IRET) instruction may not be used as the previous SS and SP/RSP values are always saved, and a stack switch is always accomplished, even if a change in CPL does not occur. Accordingly, a new instruction may be defined to accomplish a return from the SEM security exception handler.

Figs. 13A and 13B in combination form a flow chart of one embodiment of a method 1300 for providing access security for a memory used to store data arranged within multiple

memory pages. The method 1300 may, for example, reflect the exemplary rules of Table 1 for CPU-initiated (i.e., software-initiated) memory accesses when the computer system 400 of Fig. 4 is operating in the SEM. The method 1300 may be embodied within the MMU 602 (Figs. 6-7).

During a step 1302 of the method 1300, a linear address produced during execution of an instruction is received, along with a security attribute of the instruction (e.g., a CPL of a task including the instruction). The instruction resides in a first memory page. During a step 1304, the linear address is used to access at least one paged memory data structure located in the memory (e.g., a page directory and a page table) to obtain a base address of a selected memory page and security attributes of the selected memory page. The security attributes of the selected memory page may include, for example, a U/S bit and a R/W bit of a page directory entry and a U/S bit and a R/W bit of a page table entry.

During a decision step 1306, the security attribute of the instruction and the security attributes of the selected memory page are used to determine whether or not the access is authorized. If the access is authorized, the base address of the selected memory page and an offset are combined during a step 1308 to produce a physical address within the selected memory page. If the access is not authorized, a fault signal (e.g., a general protection fault signal or GPF signal) is generated during a step 1310.

During a step 1312, which may be carried out substantially at the same time as the step 1304 (i.e., substantially in parallel with the step 1304), at least one security attribute data structure located in the memory (e.g., the SAT directory 904 of Fig. 9 and a SAT) is accessed using the linear address produced during execution of the instruction to obtain an additional security attribute of the first memory page, wherein the first memory page includes the instruction, and an additional security attribute of the selected memory page, wherein the selected memory page includes the physical address corresponding to the linear address. The

additional security attribute of the first memory page may include, for example, a security context identification (SCID) value as described above, wherein the SCID value indicates a security context level of the first memory page. Similarly, the additional security attribute of the selected memory page may include an SCID value, wherein the SCID value indicates a security context level of the selected memory page.

During a step 1314, a numerical value conveyed by the additional security attribute of the first memory page is compared to a numerical value conveyed by the additional security attribute of selected memory page. The selected memory page is accessed during a step 1316 dependent upon a result of the comparing of the numerical values conveyed by the security attribute of the first memory page and the additional security attribute of selected memory page.

For example, where the method 1300 reflects the exemplary rules of Table 1 above for CPU-initiated (i.e., software-initiated) memory accesses when the computer system 400 of Fig. 4 is operating in the SEM, the selected memory page may be accessed during the step 1316 only if the integer value represented by the SCID value of the instruction is numerically less than or equal to the integer value represented by the SCID value of the selected memory page (i.e., if CEI-SCID <= SMP_SCID). If, on the other hand, the integer value represented by the SCID value of the instruction is numerically greater than the integer value represented by the SCID value of the selected memory page (i.e., if CEI-SCID > SMP_SCID), the selected memory page may not be accessed during the step 1316. In this situation, the SEM SECURITY EXCEPTION signal may be asserted as indicated in Table 1 and described above.

The particular embodiments disclosed above are illustrative only, as the invention may be modified and practiced in different but equivalent manners apparent to those skilled in the art having the benefit of the teachings herein. Furthermore, no limitations are intended

to the details of construction or design herein shown, other than as described in the claims below. It is therefore evident that the particular embodiments disclosed above may be altered or modified and all such variations are considered within the scope and spirit of the invention. Accordingly, the protection sought herein is as set forth in the claims below.